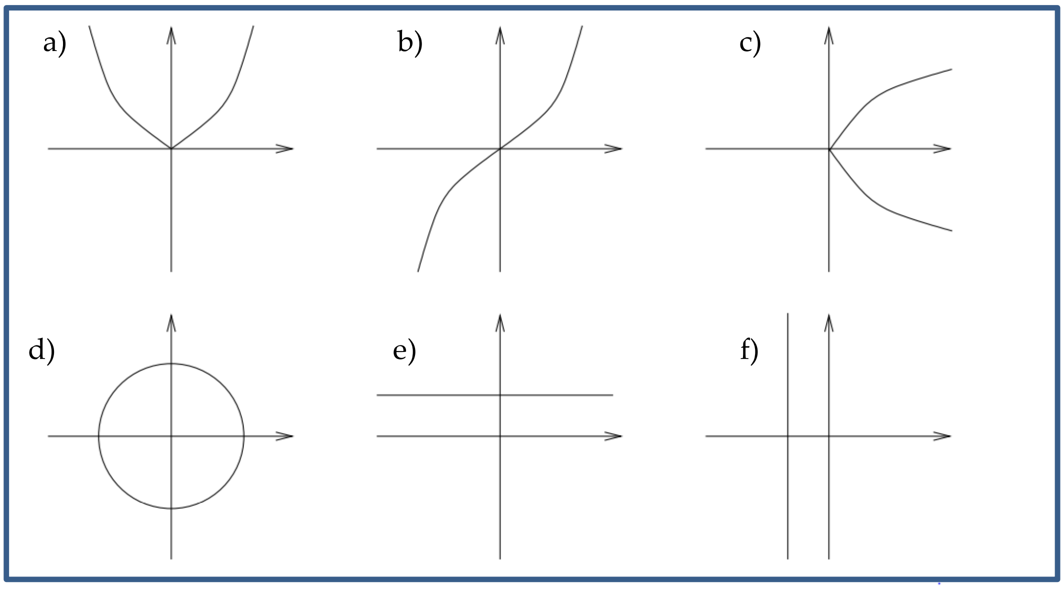
Lab 05: Algorithmic Complexity & timing code

This lab contains practical exercises related to lecture 5. Some are more about the theory while others are about Python. You should also use this lab to get help with the spell checker case study. We will leave extending it until next week.

1. Functions, Bijections and Relations

A **relation** is a set of ordered pairs (x,y). As a **function** is a set of ordered pairs where each x-element has only one y-element associated with it, all functions are relations. A **bijection** is a one-to-one function, where every element of one set is paired with exactly one element of the other set, and every element of the other set is paired with exactly one element of the first set.
That means bijections are reversible functions, i.e. y→ x is also a function.
Which of these six graphs show relations, functions, bijections?



2. Python functions and methods

Many Python methods and operators are just functions in the mathematical sense.
What are the domains and co-domains (ranges) of the functions/methods below?
If a = "abc" or a=4, what are the results? The first row has been filled in as an example.

Function / method	domain	co-domain (range)	Result when a = “abc”	Result when a=4
a+a	numbers,strings, lists, tuples	numbers,strings, lists, tuples	'abcabc'	8
len(a)				
bool(a)				
max(a)				
a.capitalize()				
a.endswith("c")				
a.find("b")				
a.isalpha()				
a.isdigit()				

3. Complexity of Algorithms 1

Consider the following three algorithms for determining whether anyone in the room has the same birthday as you.

- Algorithm 1: You say your birthday, and ask whether anyone in the room has the same birthday. If anyone does have the same birthday, they answer yes.

- Algorithm 2: You tell the first person your birthday, and ask if they have the same birthday; if they say no, you tell the second person your birthday and ask whether they have the same birthday; etc, for each person in the room.

- Algorithm 3: You only ask questions of person 1, who only asks questions of person 2, who only asks questions of person 3, etc. You tell person 1 your birthday, and ask if they have the same birthday; if they say no, you ask them to find out about person 2. Person 1 asks person 2 and tells you the answer. If it is no, you ask person 1 to find out about person 3. Person 1 asks person 2 to find out about person 3, etc.

a) What is the "problem size", i.e. the factor that affects the number of questions asked?

b) Determine the worst case, average case and best case big-theta time complexity of each of the three algorithms in terms of the number of questions asked.

c) If we change the problem to be "find out how many others in the same room have the same birthday as you", how does this affect the complexity of the algorithms?

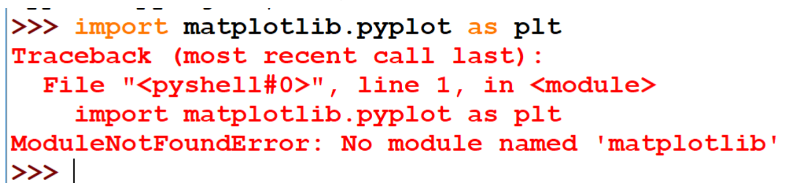
(this exercise was inspired by http://pages.cs.wisc.edu/~vernon/cs367/notes/3.COMPLEXITY.html)

4. Pip install: installing non-standard packages

For later exercises you need numpy and pyplot (part of matplotlib).
In IDLE, try

```
>>> import matplotlib.pyplot as plt
```

You should see the following error:



```
>>> import matplotlib.pyplot as plt
Traceback (most recent call last):
  File "<pyshell#0>", line 1, in <module>
    import matplotlib.pyplot as plt
ModuleNotFoundError: No module named 'matplotlib'
>>> |
```

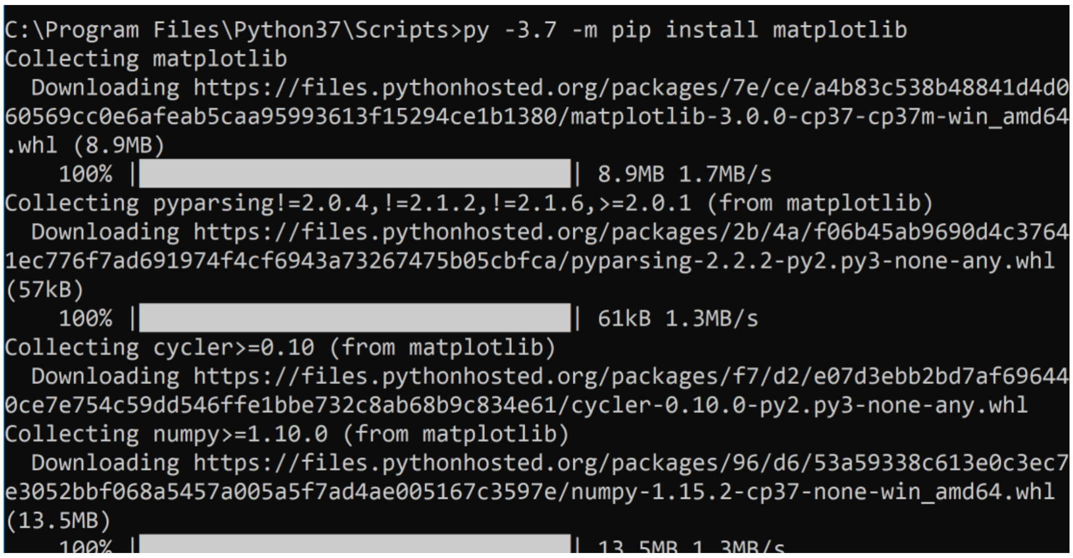
Q: What is the reason for this error?

Open a Windows Command Line as administrator (right-click and choose the appropriate option to do this). Use pip install to install **matplotlib**. The command you need is

```
py -3.7 -m pip install matplotlib
```

NOTE: pip is in the Scripts subfolder of the Python folder, Python37 (In my case the directory is C:\Program Files\Python37\Scripts). If this is not in your path, you will need to cd to this folder to be able to use pip.

This should look like the screenshot below. Matplotlib has several components, so be patient and wait until the install is completed successfully.



```
C:\Program Files\Python37\Scripts>py -3.7 -m pip install matplotlib
Collecting matplotlib
  Downloading https://files.pythonhosted.org/packages/7e/ce/a4b83c538b48841d4d0
60569cc0e6afeab5caa95993613f15294ce1b1380/matplotlib-3.0.0-cp37-cp37m-win_amd64
.whl (8.9MB)
    100% |████████████████████████████████| 8.9MB 1.7MB/s
Collecting pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 (from matplotlib)
  Downloading https://files.pythonhosted.org/packages/2b/4a/f06b45ab9690d4c3764
1ec776f7ad691974f4cf6943a73267475b05cbfca/pyparsing-2.2.2-py2.py3-none-any.whl
(57kB)
    100% |████████████████████████████████| 61kB 1.3MB/s
Collecting cycler>=0.10 (from matplotlib)
  Downloading https://files.pythonhosted.org/packages/f7/d2/e07d3ebb2bd7af69644
0ce7e754c59dd546ffe1bbe732c8ab68b9c834e61/cycler-0.10.0-py2.py3-none-any.whl
Collecting numpy>=1.10.0 (from matplotlib)
  Downloading https://files.pythonhosted.org/packages/96/d6/53a59338c613e0c3ec7
e3052bbf068a5457a005a5f7ad4ae005167c3597e/numpy-1.15.2-cp37-none-win_amd64.whl
(13.5MB)
    100% |████████████████████████████████| 13.5MB 1.3MB/s
```

Go back to IDLE and again

```
>>> import matplotlib.pyplot as plt
```

The command should work this time!

Familiarise yourself with the plot function that is part of pyplot by checking for built-in help:

```
>>> help(plt.plot)
>>> help(plt.plot)
Help on function plot in module matplotlib.pyplot:

plot(*args, scalex=True, scaley=True, data=None, **kwargs)
    Plot y versus x as lines and/or markers.

    Call signatures::

        plot([x], y, [fmt], data=None, **kwargs)
        plot([x], y, [fmt], [x2], y2, [fmt2], ..., **kwargs)

    The coordinates of the points or line nodes are given by *x*, *y*.

    The optional parameter *fmt* is a convenient way for defining basic
    formatting like color, marker and linestyle. It's a shortcut string
    notation described in the *Notes* section below.

    >>> plot(x, y)        # plot x and y using default line style and color
    >>> plot(x, y, 'bo')  # plot x and y using blue circle markers
    >>> plot(y)           # plot y using x as index array 0..N-1
    >>> plot(y, 'r+')     # ditto, but with red plusses
```

5. Plotting with pyplot

In the lecture, you saw two scripts showing examples of plotting with pyplot (**pyplot_ex1.py** and **pyplot_numpy.py**). Run these examples.

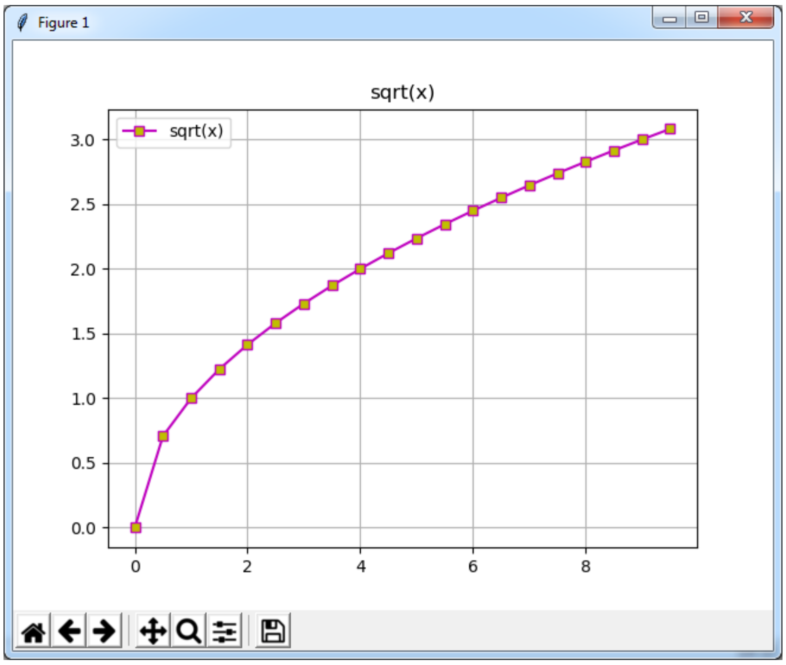
a) To make sure you understand the code, experiment with it - change some parameters and see what happens.

For the tasks below, you will have to experiment and look up matplotlib/pyplot documentation. (Remember to use the American spelling "color".)

b) Adapt the code in order to plot f(x)=sqrt(x) for x between 0 and 10 in intervals of 0.5.

c) Adapt this code further so that your plot from b) is shown with a magenta line.

d) Adapt this code further so that your plot is shown as below, with a magenta line and square markers that are filled in yellow, as well as a title, a legend, and gridlines. (You will have to experiment and look up matplotlib/pyplot documentation. Remember to use the American spelling "color".)



6. Generating large lists of random numbers

To really compare sorting algorithms, you need to generate long lists of random numbers multiple times, so it's worth finding out efficient ways of doing this.
In this week's materials on moodle, you will find **large_list.py**, which shows you 3 ways of doing this and runs a timing comparison.
Run this script, then inspect the code. Make sure you understand the code.

Q: Are there noticeable differences in the runtime of each method?

Q: Which method is the most efficient?

(Note: if you search for answers to this problem on the internet, you may come across an xrange() function. This has now been renamed to range() to make Python more efficient.).

7. Adding up the integers from 1 to N

From scratch, write a Python function that

- *takes N as a parameter*
- *adds up the numbers from 1 to N*
- *And returns the result*

So e.g. if N = 5 we want the calculation 1+2+3+4+5 = 15
There are at least 4 approaches I can think of. *(you need around 1-4 lines within the function)*

Write the function on paper or in a text editor.
Give yourself exactly 3 minutes to do this – you are welcome to work with a colleague, but do not use the internet for help.
There is no need to test your function at this point.

DO NOT READ THE FOLLOWING EXERCISE UNTIL YOU HAVE COMPLETED THIS ONE

8. Adding up the integers from 1 to N (part 2)

Open the file timing_test1.py (from moodle)

DO NOT RUN THE SCRIPT JUST YET

Compare your answer from exercise 7 above with the four functions (loopy, listy, pythonic, mathematical) in the script.

Q: Which of the four functions is yours similar to?

Q: Why did you write your function this way?

Q: Which of the four functions do you think is best?

Q: Why? (Use your knowledge about algorithmic complexity to explain and justify your answer)

Q: Which of the functions do you think is/are the worst in terms or efficiency? Why?

9. Adding up the integers from 1 to N (part 3)

Now run timing_test1.py without modifying it. It should have N = 10,000.

Q: Which of the functions is fastest? Which is slowest?

Q: Do the results match your expectations? (if not, what is different?)

Run timing_test1.py 4 more times without making any modifications. Then run it for N=100 and N=1,000,000.

Compare the results.

Q: Are the timing results the same for all 5 runs, or do they vary?

Q: If they vary, is the relative performance of the functions consistent (i.e. is the worst function always the same?)

10. Putting it all together

Now modify timing_test.py as follows.

- Write a function main(). Move the code from the boiler plate into this function and in the boiler plate call main().
- Test that your script still works as before. Use N=1000 for this purpose.
- Modify the script so that every one of the four functions is run once each with N=100, N=1000, N=10000, N=100000, N=1000000. In addition to printing, the results need to be stored in variables – you could use lists, tuples or dictionaries for this purpose. (For easy plotting, lists may be best)
- Test that your script runs

Finally, add code to plot your results (y=time against x=N) using pyplot. All four functions should be plotted on the same graph.

OPTIONAL EXTENSION EXERCISE BELOW

11. Exponential growth

Water lilies are planted in a new pond, and their growth is observed.
The pond is $40m^2$.
On day 0, a quarter of the pond surface (s), $10m^2$, is covered.

Q: When will the whole surface be covered if growth is exponential, doubling every day?

Hint: You can work this out without a formula – it's a "typical" puzzle question.

Now let's make it more tricky:
When will the whole surface be covered if growth is exponential

a) increasing by 50% every day?
b) increasing by 10% every day?

For this you need the formula for exponential growth, which is $\mathbf{x_t = x_0(1+r)^t}$.
Here x_t is the value on day t, x_0 the value on day 0, and r the growth rate (e.g. 50% is a growth rate of 0.5).

To theoretically solve this equation for t, you can take logs of both sides of the equation and then use the fact that $\ln(a^t)=t \ln(a)$.
Logs may seem a bit weird but they have many uses, and they were invented right here, in Merchiston Tower, by John Napier, our University's namesake.

As this is a scripting module, **write a script** to calculate the surface area covered each day.
Use a WHILE loop to stop once the amount covered reaches the whole surface area (s).
Remember to use variables for x_0, r and s.

Run your script with appropriate values for a) and b)

Q: what are the answers (approximately) for question a) and b)?